

# A measure of compression gain for new alphabet symbols in data-compression

RICHARD FREDLUND

richard\_fredlund@hotmail.com

## Abstract

*In coding theory it is widely known that the optimal encoding for a given alphabet of symbol codes is the Shannon entropy times the number of symbols to be encoded. However, depending on the structure of the message to be encoded it is possible to beat this optimal by including only frequently occurring aggregates of symbols from the base alphabet. We prove that the change in compressed message length by the introduction of a new aggregate symbol can be expressed as the difference of two entropies, dependent only on the probabilities of the characters within the aggregate plus a correction term which involves only the probability and length of the introduced symbol. The expression is independent of the probability of all other symbols in the alphabet. This measure of information gain, for a new symbol, can be applied in data compression methods.*

## I. INTRODUCTION

The symbol alphabet used for most data compression schemes are either individual characters, syllables (for example [1, 2]), or words (for example [3, 4]). With a few technicalities (e.g. punctuation in word encodings) these alphabets essentially form a partition of the text into more or less, similar parts. Spaces have been added to words [5] and frequently occurring 2,3 and 4 grams have also been used to pad out the alphabet, [6]. However, adding sequences of arbitrary length to a base alphabet of characters, based on information gain, to be used by a variable length compression scheme such as Arithmetic coding [7] or Huffman encoding [8] has not been considered.

Shannon's theorem [9] states that the best (per character) compression achievable for a message (and closely approximated by Arithmetic encoding [7, 10]) is its entropy relative to the alphabet. At first glance adding symbols to a base alphabet in order to further reduce this rate of compression may seem counter to Shannon's theorem. However, Shannon's theorem is true only for a fixed alphabet, and there is no reason why, depend-

ing on the structure of the message, adding aggregate symbols may not decrease the compression further.

We ask the question at what point is an aggregate symbol, composed of two or more alphabet symbols worth adding to the model. For example when encoding simple text, composed of 26 alphabet symbols at what frequency is the compound symbol 'the' composed of three alphabet symbols worth adding to the model? To realize that for high enough frequencies adding the symbol may produce a decrease in the expected message length notice that the number of 'symbols' in the message will decrease by two times the number of occurrences of the word 'the'. As each occurrence of the word 'the' represents a condensing of three alphabet symbols into just one new symbol.

We define the gain of an new aggregate symbol to be the decrease in optimal encoding length with the introduction of the symbol. However, with the addition of a new aggregate symbol it is also possible that the optimal encoding length of the message with respect to the alphabet will increase, rather than de-

crease. The gain therefore may be positive or negative.

In Section II we prove our main result, that the average gain, as measured in bits per character, of introducing a new symbol  $S$  can be expressed as the difference of two entropies, each expressed only in terms of the relative frequencies of the alphabet symbols, which are being combined to form  $S$  plus a correction term which involves only the probability and length of  $S$ .

In Section III, we perform an experiment to demonstrate the use of the information gain measure to create an aggregate alphabet for data compression. Starting with a base alphabet of just 72 distinct characters and an optimal compression rate of 4.5 bits per character, we discover an alphabet with 1319 aggregate symbols which effectively reduces the rate of compression to under 3 bits per character.

## II. MAIN RESULT

Notation: start by considering a message  $M = m_1, m_2, \dots, m_N$  of  $N$  character symbols from the alphabet  $A_x = \{a_1, a_2, \dots, a_K\}$  with known character frequencies  $F_x = \{f_{a_1}, f_{a_2}, \dots, f_{a_K}\}$ . The probabilities of each character symbol are  $P_x = \{p_{a_1}, p_{a_2}, \dots, p_{a_K}\}$  where  $p_{a_i} = \frac{f_{a_i}}{N}$  for each  $i$  in  $\{1, 2, \dots, K\}$ .

### II.1 Theorem

We prove that when a new aggregate symbol  $S = b_1 b_2 \dots b_r$  composed of  $r$  (not necessarily distinct) symbols from the original alphabet  $A_x$  is added to the alphabet and  $\{S\}$  represents the set of distinct characters in  $S$ , then the per character information gain can be expressed as the difference of two entropies:

$$\sum_{p_{b_i} \in \{S\}} p_{b_i} \log_2 \frac{1}{p_{b_i}} - \sum_{p_{b_i} \in \{S\}} \lambda_{b_i} \log_2 \frac{1}{\lambda_{b_i}} \quad (1)$$

(Where  $\lambda_{b_i}$  represents the probability of symbol  $b_i$  after occurrences of  $b_i$  in  $S$  have been removed) plus a correction term expressed only

in terms of  $p_s$  the probability of  $S$  in the message, and  $r$ .

Letting  $\mu_s = (1 - p_s(r - 1))$ , the correction term is again the difference of two entropies and is given by:

$$\mu_s \log_2 \frac{1}{\mu_s} - p_s \log_2 \frac{1}{p_s} \quad (2)$$

We call  $\mu_s$  the rescale constant, and the length of the new message  $N'$  is  $\mu_s N$ .

### II.2 Proof

By Shannon's information theory [9] the optimal encoding for this message gives an average code-length for each encoded symbol of:

$$L(C, X) \approx H(X) = \sum_{x \in A_x} p_x \log_2 \frac{1}{p_x} \quad (3)$$

where  $H(X)$  is the entropy of the message  $M$  for this probability model, described by  $(A_x, P_x)$ . The optimal compressed code length for the whole message  $M$  is the message length  $N$  times this entropy:

$$L(M, A_x) = N \times H(x). \quad (4)$$

This can be re-written in terms of the character frequencies as:

$$L(M, A_x) = N \log_2 N - \sum_{x \in A_x} f_x \log_2 f_x \quad (5)$$

using the identity  $\log_2 \frac{1}{x} = -\log_2 x$  and by noting that the sum of the frequencies  $\sum_{x \in A_x} f_x$  is equal to  $N$ .

Let  $S = b_1 b_2 \dots b_r$  be an aggregate symbol composed of  $r$  symbols from the original alphabet  $A_x$  so that  $b_i \in A_x$  for  $i$  in  $\{1, 2, \dots, K\}$ , (Note that  $b_i$  and  $b_j$  may not be distinct as  $S$  may contain more than one copy of some characters in  $A_x$ ), and let  $f_S$  be the number of non-overlapping occurrences of  $S$  in the message  $M$ .

Adding  $S$  to the alphabet gives a new alphabet of character symbols,  $A'_x = \{S, a_1, a_2, \dots, a_K\}$ , a new set of probabilities  $P'_k = \{p'_s, p'_{a_1}, \dots, p'_{a_K}\}$  and a new set of character frequencies  $F'_k = \{f'_s, f'_{a_1}, \dots, f'_{a_K}\}$ .

The message length  $N$  will also change. This is because, for each of the  $f_S$  occurrences of  $S$  in the message,  $r$  symbols from the original character sequence  $M$  will be condensed into just one character in the new alphabet  $A'_x$ . This gives a new message length of:

$$N' = N - f_S(r - 1) \quad (6)$$

By counting the number of characters  $a_i$  in the original message and subtracting occurrences which appear in  $S$  we get the updated character frequencies:

$$f'_{a_i} = f_{a_i} - f_S S(a_i) \quad (7)$$

where  $S(a_i)$  is the number of times the character symbol  $a_i$  appears in  $S$ . Note for characters not in  $S$  the number of occurrences will remain unchanged and  $f'_{a_i} = f_{a_i}$ .

Substituting 6 and 7 into 5 gives the new expected code length for the message, with the introduction of the new aggregate symbol  $S$ .

$$\begin{aligned} L(M', A'_x) = & N' \log_2 N' - f_S \log_2 f_S \\ & - \sum_{a_i \notin S} f_{a_i} \log_2 f_{a_i} \\ & - \sum_{a_i \in S} f'_{a_i} \log_2 f'_{a_i} \end{aligned} \quad (8)$$

Here  $M'$  is the message written in the new alphabet  $A'_x$  and  $L(M', A'_x)$  is the new expected coding length for the message.

We have split the summation of the log frequencies into three distinct parts. Firstly, the term containing  $f_S$  the number of occurrences of the newly introduced character  $S$ . Secondly, those characters not in  $S$  whose frequency do not change with the introduction of  $S$  and finally, those characters which are in  $S$  and whose frequencies do change.

The advantage of this formulation is that the terms involving frequencies not in  $S$  cancel and comparison of 5 and 8 leads to a formula, in terms of the symbol frequencies, for the expected coding gain  $gain(S)$  over the whole

message of introducing  $S$  into the alphabet:

$$\begin{aligned} gain(S) = & N \log_2 N - N' \log_2 N' \\ & + f_S \log_2 f_S \\ & - \sum_{a_i \in S} f_{a_i} \log_2 f_{a_i} \\ & + \sum_{a_i \in S} f'_{a_i} \log_2 f'_{a_i} \end{aligned} \quad (9)$$

The addition of the aggregate symbol  $S$  to the alphabet  $A_x$  will shorten the optimally encoded message if and only if  $gain(S) > 0$ .

The disadvantage of this formulation is that the expected information gain per symbol, (which is  $\frac{gain(S)}{N}$ ), appears to be dependent on  $N$ , due to the  $\log_2 N$  term. To see that this ought not to be the case consider the scenario of a new message  $M_{new}$  which is precisely  $M$  repeated twice, such that all character frequencies are exactly doubled. We would expect to see the overall gain for this new message to be  $2 \times gain(S)$ .

We now recast 9 in terms of probabilities. Starting by replacing  $f_S$  with  $Np_S$  in 6 to give  $N' = N\mu_S$ , where  $\mu_S = (1 - p_S(r - 1))$ , which also give  $f'_{a_i} = p'_{a_i}\mu_S N$ . Substituting these in 9 gives:

$$\begin{aligned} gain(S) = & N \log_2 N - \mu_S N \log_2 \mu_S N \\ & + p_S N \log_2 p_S N \\ & - \sum_{a_i \in S} p_{a_i} N \log_2 p_{a_i} N \\ & + \sum_{a_i \in S} p'_{a_i} \mu_S N \log_2 p'_{a_i} \mu_S N \end{aligned} \quad (10)$$

This simplifies by noticing that all terms are of the form  $AN \log_2 AN$  and using the identity  $AN \log_2 AN = AN \log_2 N - NA \log_2 \frac{1}{A}$ . Summing up the  $N \log_2 N$  terms everything cancels by noting that  $p_S \times r = \sum_{a_i \in S} p_S S(a_i)$ . This gives a sum over terms all of the form  $A \log_2 A$ , the first of which is  $1 \log_2 1 = 0$  which also cancels to give:

$$\begin{aligned}
\frac{\text{gain}(S)}{N} &= \mu_s \log_2 \frac{1}{\mu_s} - p_s \log_2 \frac{1}{p_s} \\
&+ \sum_{a_i \in S} p_{a_i} \log_2 \frac{1}{p_{a_i}} \\
&- \sum_{a_i \in S} p'_{a_i} \mu_s \log_2 \frac{1}{p'_{a_i} \mu_s}
\end{aligned} \tag{11}$$

We finish by noting that  $p'_{a_i} \mu_s = \frac{f_{a_i} - f_{S^c}(a_i)}{N} = p_{a_i} - p_{S^c}(a_i)$  and let  $\lambda_{a_i} = p_{a_i} - p_{S^c}(a_i)$ . The average information gain per character symbol can therefore be written as:

$$\begin{aligned}
\text{char\_gain}(S) &= \sum_{p_{b_i} \in \{S\}} p_{b_i} \log_2 \frac{1}{p_{b_i}} \\
&- \sum_{p_{b_i} \in \{S\}} \lambda_{b_i} \log_2 \frac{1}{\lambda_{b_i}} \\
&+ \mu_s \log_2 \frac{1}{\mu_s} - p_s \log_2 \frac{1}{p_s}
\end{aligned} \tag{12}$$

which is independent of the probabilities, of all other alphabet symbols except those contained in  $S$  and the probability of  $S$  itself.

In more concise notation this can be re-written as:

$$\begin{aligned}
\text{char\_gain}(S) &= H(\mu_s, p_{a_i} : a_i \in \{S\}) \\
&- H(p_s, \lambda_{a_i} : a_i \in \{S\})
\end{aligned} \tag{13}$$

### III. EXPERIMENT

Here we perform an experiment to demonstrate the use of the information gain measure to create an aggregate symbol alphabet for lossless text compression by finding an alphabet of aggregate symbols for a single document, *Alice in wonderland* (*alice29.txt*) from the Canterbury text corpus.

While both encoder and decoder are assumed to have access to a small basic alphabet of characters, without the need for transmission with the message, all other characters, (from the character set, and from the aggregate symbols) must also be encoded in the bit-stream for transmission. We therefore make

a distinction between the message encoding, and the alphabet encoding. For each new symbol the alphabet encoding will increase in size, so the message encoding must decrease by at least this amount to be worth adding to the alphabet.

#### III.1 Alphabet encoding scheme

We start with a very basic standard alphabet of just 64 characters, which are assumed to be known previously, by both the encoding and decoding software. This character set is simply the uppercase and lowercase letters, and 12 punctuation symbols. These are, space, newline, colon, semi-colon, exclamation mark, open and close brackets, hyphen, comma, full-stop and question mark.

The first 5 bits of the alphabet encoding are used to encode, *MXBITS*, the number of bits required to encode a frequency count in this document. The maximum being 32 bits assigned to each frequency count, which would allow up to 4294 million occurrences of any character within the document. *MXBITS* is determined by the maximum number of occurrences of any character in the document. For *Alice29.txt*, *MXBITS* is set to 15 indicating that 15 bits should be used to record the frequency of each alphabet symbol. (Note that in practice this could be considerably reduced by noting that this is only the maximum ever required, and in practice most symbol frequencies for this document fall into the range 0-255 and could be expressed using only 8 bits).

Next each of the 64 characters from the standard alphabet is assigned 15 bits to record it's frequency in the document (960 bits).

The next 8 bits in the encoding are assigned to *NUMCHAR*, the number of non-standard characters in the document. We keep things simple here and assume all other characters in the document are 8 bit ascii codes (unicode symbols are not considered).

In *Alice.txt* there are 8 characters not included in the standard alphabet. These are apostrophe, two forms of speech-mark, \*, open and close, square bracket, and the nu-

merals, 2 and 9.

These are encoded with 8 bits for ascii code, and 15 bits for character frequency. That is  $23 * 8 = 184$  bits.

Before any aggregate symbols are added therefore, the number of bits, used by the alphabet encoding are  $5 + 960 + 8 + 184 = 1157$ .

The next 15 bits of the encoding are allocated to AGCOUNT, the number of aggregate symbols to be included in the alphabet.

At initialization, AGCOUNT is set to 0 and the alphabet contains  $64 + 8 = 72$  characters. Aggregate symbols in this implementation consist of two ordered symbols, from the already existing alphabet. As new aggregates are added, they may consist of combinations of previous aggregates, or of characters from the base alphabet. As the aggregate alphabet increases in size, the number of possible aggregates also increases. For an alphabet of length  $N$ , the first and second symbol both have  $N$  possible values giving a total of  $N^2$  possible new aggregate symbols. The number of bits required to encode each aggregate symbols depends on the number of bits required to encode a value in  $\{1, \dots, N\}$ . At initialization  $N = 72$ , hence the number of bits to encode each sub-symbol in the aggregate is given by  $\lceil \log_2(AGCOUNT + 72) \rceil$ . 15 bits must also be allocated to record the frequency count. This gives a total of  $15 + 2\lceil \log_2(AGCOUNT + 72) \rceil$  bits for each new aggregate symbol. This starts of at 29 bits and increases by 2 bits each time  $N$  gets too large.

### III.2 Search algorithm

Here we have kept the search for aggregate symbols with maximum, compression gain as simple as possible. At each step the symbol gain is calculated for each of the possible  $N^2$  aggregates and the highest gain is selected. If this gain is higher than the number of bits necessary for the alphabet encoding, then the symbol is added.

This approach becomes infeasible as  $N$  grows large, and it is expected that better search algorithms can be found to

find aggregates with a high expected gain.

	gain	symbol	freq
1	6538	he	3705
2	4694	the	2101
3	3249	in	1998
4	4051	ing	979
5	3098	ou	1515
6	3054	[sp] the	1838
7	2135	, [sp]	1761
8	2020	[nl][nl]	827
9	2167	. [nl][nl]	404
10	2064	ve	663
11	1968	nd	1172
12	2648	and	880
13	1899	, '	397
14	1995	' [nl][nl]	343
15	1792	, [sp] and	418
16	1693	-	262
17	1553	[sp]the[sp]	1314
18	1392	th	1096
19	1345	Al	403
20	1832	Ali	395

### III.3 Results

The brute force algorithm described above was implemented on Alice29.txt from the Canterbury Corpus. The uncompressed document contains 148480, 8 bit ascii characters. At initialisation there were 72 distinct characters in the base alphabet. The alphabet encoding occupied 1157 bits and the optimal message encoding length which could be closely approximated by arithmetic encoding was 670057 bits. So the total message encoding before any aggregates were introduced was 671214 bits. This gives a message encoding of 4.51 (2.d.p.) bits per character. and a total encoding of 4.52 (2.d.p.) bits per character. Subsequently 1319 aggregate symbols were added to the alphabet. This increased the number of bits required by the alphabet encoding to 46674 bits. However, the optimal message encoding dropped to 390937 bits, leaving a total encoding length of 437611 bits. This gives a message encoding of 2.63 (2.d.p.) bits per character and a total encoding of 2.95(2.d.p) bits per character.

The bit rate could have dropped further

however by this point the experiment had slowed down considerably, so the experiment was stopped soon after the bit rate of 3 bits per character was reached.

The above table shows the first 20 aggregate symbols selected by the algorithm. Notice that the first 18 symbols seem quite generic and might well be informative symbols for many text documents. It is interesting to note that the second symbol is 'the' while 'th' is not selected until the 18th symbol, instead 'he' is selected as the first symbol, having high frequency and an information gain of 6538 bits.

However, symbol 19 and 20 are 'Al' and 'Ali' and the next two symbols are 'Alic' and 'Alice' respectively. These symbols are specific to the document. Other aggregate symbols, which are recognizable as being specific to Alice29.txt and appear in the alphabet are, 'Dormouse', 'Hare', 'Mock', 'curious', 'remark', 'said Alice', 'little', 'the Queen' and 'Duchess'.

Notice that the information gain is not always strictly decreasing with each new aggregate symbol. The information gain of 'ing', 'and' and 'Ali' are each higher than their immediate predecessors, 'in', 'nd' and 'Al', respectively. Meaning that where a longer aggregate symbol has high gain, the shorter aggregates which build up to the longer symbol, may have a lower gain.

### III.4 Discussion of Results

The aggregate alphabet found, achieves a 41.7% (1.d.p.) reduction in message encoding length, and a 34.7% (1.d.p.) reduction in total encoding length relative to the Shannon optimal for the basic alphabet of single characters. Reaching a message encoding of 2.63 (2.d.p.) bits per character and a total encoding of 2.95 (2.d.p) bits per character before the experiment was stopped.

Both the alphabet encoding, and the search algorithm used to find new aggregate symbols, are not ideal, and leave considerable scope to be improved upon, in developing a

practical data compression scheme. However, the results obtained demonstrate the basic efficacy of using this measure of information gain to create an alphabet for data compression.

The two drawbacks of this implementation are: firstly, it can only find larger chunks which are informative by working up through smaller chunks first. e.g. 'al', 'ali', 'alic', 'alice'. (which requires the smaller chunks to also be informative); secondly, once a larger chunk is found, the smaller chunks which lead up to it, will often become less informative in the alphabet. However it is interesting that the pairwise-encoder is able to find some longer sequences such as 'the Queen', and 'said Alice', showing there is enough information gain in the pairwise steps to lead up to these particular sequences.

Better compression rates could be achieved by using a dictionary approach and initially adding whole words before starting the pairwise combination of symbols, and also removing smaller chunks if they become uninformative.

## IV. CONCLUSION

The contribution in Section II, has been to find a simple formulation which measures the change in compression length when adding a new aggregate symbol to an alphabet code for data compression. This formulation is concise and independent of the frequency of all other characters from the alphabet not in the new symbol. Computation time for this measure is therefore independent of size of the alphabet.

This measure is potentially applicable to any variable length coding scheme with an alphabet and could be used with either static or adaptive schemes such as PPM [10].

One advantage of using this measure to create alphabets for symbol codes is that no prior knowledge of the structure of the data is required, and like Lempel-Ziv [11] it can be applied to any data which contains repetitions.

---

## REFERENCES

- [1] J. Lansky and M. Zemlicka, "Compression of small text files using syllables," in *DCC*, p. 458, IEEE Computer Society, 2006.
- [2] I. Akman, H. Bayindir, S. Ozleme, Z. Akin, and S. Misra, "A lossless text compression technique using syllable based morphology," *Int. Arab J. Inf. Technol.*, vol. 8, no. 1, pp. 66–74, 2011.
- [3] A. Moffat and R. Y. K. Isal, "Word-based text compression using the burrows-wheeler transform," *Inf. Process. Manage.*, vol. 41, no. 5, pp. 1175–1192, 2005.
- [4] A. Farina, G. Navarro, and J. R. Parama, "Boosting text compression with word-based statistical encoding," *Comput. J.*, vol. 55, no. 1, pp. 111–131, 2012.
- [5] R. Y. K. Isal, A. Moffat, and A. C. H. Ngai, "Enhanced word-based block-sorting text compression," in *ACSC* (M. J. Oudshoorn, ed.), vol. 4 of *CRPIT*, pp. 129–137, Australian Computer Society, 2002.
- [6] S. Grabowski, "Text processing for burrows-wheeler block sorting compression," in *VII Konferencja Sieci i systemy Informatyczne-Teoria, Projekty, Wdrozenia, Lodzkiej*, 1999.
- [7] F. Rubin, "Arithmetic stream coding using fixed precision registers," *IEEE Transactions on Information Theory*, vol. 25, no. 6, pp. 672–675, 1979.
- [8] D. Huffman, "A.: A method for the construction of minimum redundancy codes," in *Proceedings of the IRE*, 1952.
- [9] D. MacKay, *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2006.
- [10] P. G. Howard and J. S. Vitter, "Analysis of arithmetic coding for data compression," in *Data Compression Conference* (J. A. Storer and J. H. Reif, eds.), pp. 3–12, IEEE Computer Society, 1991.
- [11] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inf. Theor.*, vol. 24, pp. 530–536, Sept. 2006.